Windows NT®

Operating System

# Microsoft Windows NT – The Foundation

## White Paper

**Abstract**

This paper explains the primary design goals of Microsoft® Windows NT® – robustness, extensibility and maintainability – and explains how these attributes informed and directed each aspect of the initial system design and its development over the last several years. Windows NT was designed to provide a solid foundation for future development efforts by Microsoft and the larger development community. This paper describes the original design process, provides a high-level overview of the architecture, and concludes with a real-world example of the coding standards implemented and maintained by the Windows NT design team.

## CONTENTS

## INTRODUCTION

From its inception, Microsoft® Windows NT® was designed to be a robust, portable operating system that would be maintainable, flexible, and secure over time. This paper describes the design decisions that were made during the initial planning of Windows NT, describes each of the design goals in priority order (as documented in the original design documentation[1]), and describes how these goals were met in the system architecture. Finally, the paper provides an example of a subsystem component specifically designed to meet the goals of robustness and maintainability over time.

### The Mission

When the Windows NT development team was formed in 1989, it had a clear mission: *to design and build a personal computer operating system that would meet the current and future operating system needs of the PC platform*. To meet this objective, the design team identified the following market requirements:

- To provide easy portability to other 32-bit architectures.
- To provide scalability and multiprocessing support.
- To support distributed computing, allowing multiple computers to share resources.
- To support the application programming interfaces (APIs) required by POSIX.
- To provide U.S. government Class 2 (C2) security features, and to provide a path to Class B1 and beyond.

### The Design Goals

Based on market requirements and Microsoft's development strategy, the original Microsoft NT design team established the a set of prioritized goals. Note that from the outset, the priority design objectives of Windows NT were *robustness* and *extensibility*:

1. ***Robustness***. The operating system must actively protect itself from internal malfunction and external damage (whether accidental or deliberate), and must respond predictably to software and hardware errors. The system must be straightforward in its architecture and coding practices, and interfaces and behavior must be well-specified.
2. ***Extensibility and maintainability***. Windows NT must be designed with the future in mind. It must grow to meet the future needs of original equipment manufacturers (OEMs) and of Microsoft. And the system must be designed for maintainability—it must accommodate changes and additions to the API sets it supports and the APIs should not employ flags or other devices that drastically alter their functionality.
3. ***Portability***. The system architecture must be able to function on a number of platforms with minimal re-coding.

[1] "NT OS/2 Subsystem Design Rationale," June 1, 1989.

4. ***Performance***. Algorithms and data structures that lead to a high level of performance and that provide the flexibility needed to achieve our other goals must be incorporated into the design.
5. ***POSIX compliance and government certifiable C2 security***. The POSIX standard calls for operating system vendors to  implement UNIX-style interfaces so that applications can be moved easily from one system to another. U.S. government security guidelines specify certain protections such as auditing capabilities, access detection, per-user resource quotas, and resource protection. Inclusion of these features would allow Windows NT to be used in government operations.

## Design Alternatives

With its primary goals in mind, the development team investigated several alternatives during the design phase.

The first design layered the POSIX API set over a slightly extended OS/2 API set. (Originally, the operating system was to have an OS/2-style user interface and was to provide the OS/2 API as its primary programming interface. However, due largely to the greater popularity of Microsoft Windows, Microsoft refocused its strategy and developed the Win32 API, a 32-bit programming interface for development of next-generation applications.) As the design progressed, it became clear that it would result in a system that would not be robust, easily maintained, or extensible. A similar attempt during the development of OS/2 led to considerable change in the base system capabilities, which further strengthened the team's conclusion that this was a poor alternative.

The next design implemented both OS/2 and POSIX API sets directly in the Windows NT executive. This was an improvement on the previous design, but the large number of oddly structured and tricky interfaces required by this design threatened the goals of extensibility and maintainability.

The third design implemented OS/2 and POSIX as protected subsystems outside the Windows NT executive. This type of client/server architecture had been successful in the academic community and at other research sites, largely because it decoupled the more volatile services from the operating system kernel—thus preserving the integrity of the operating system while allowing system services to grow and change as necessary. After analysis and an extended mock up and test cycle, it became clear that this design would provide  the robustness, extensibility, maintainability, portability, and performance that the new operating system required.

The next section of this document provides an overview of Windows NT architecture, particularly as it relates to the crucial design goals of system robustness and maintainability over time.

## THE WINDOWS NT DESIGN

The Windows NT system design consists of a highly functional executive, which runs in privileged processor (or *kernel*) mode and provides system services and internal processes, and a set of nonprivileged servers called *protected subsystems,* which run in nonprivileged (or *user*) mode outside of the executive. Note that the executive provides the only entry point into the system—there are no back door entry points that could compromise security or damage the system in any way.

A protected subsystem executes in user mode as a regular (native) process. The subsystem may have extended privileges as compared to an application, but it is not considered a part of the executive and, therefore, cannot bypass the system security architecture or corrupt the system in any other way. Subsystems communicate with their clients and each other using high-performance local procedure calls, or LPCs.

The NT executive includes a set of system service components—the Object Manager, the Security Reference Monitor, the Process Manager, and so forth—which are exposed through a set of API-like system services. While the executive performs some internal routines, it is primarily responsible for taking an existing process thread from a requesting subsystem or application, *validating that the thread should be processed,* executing it, and then returning control of the thread to the requestor.

### Maintainability and Extensibility Over Time

The following efforts ensure that Windows NT meets its goals of maintainability and extensibility:

- The original developers of Windows NT designed the system to be simple and provided extensive code documentation. This, coupled with a common coding standard used throughout the system, has enabled subsequent programmers to work on any piece of the system without having to consult a system expert to learn about hidden rules, side effects, magical programming tricks, or Windows NT folklore. The code is straightforward—as is the documentation.
- By using subsystems to implement major portions of the system, Windows NT isolates and controls dependencies. For example, the only piece of the system affected by a change to the POSIX standard is the POSIX subsystem. The design of the process structure, memory management, synchronization primitives, and so on, are not affected.
- The Windows NT design accommodates change and growth. Subsystems that provide additional functionality can be added to the system without impacting the base system. New subsystems can be added without modifications to the Windows NT executive;  for example, new subsystems can be added to allow limited support for operating system environments other than the Microsoft-provided MS DOS, OS/2, Win32, and POSIX environments. Moreover, the executive itself is modular in design—because its components are independent from each other and interact in predictable ways, and because the interfaces between these components are so carefully controlled, it is possible to replace

a component without adversely impacting the system. If the new version implements all of the system services and internal interfaces correctly, the operating system will function as before.

- Perhaps most importantly, all subsystems can be coded to take advantage of the security features provided in Windows NT.

The following illustrates the basic architecture of the operating system through version 3.51 (note that only a few of the subsystems are illustrated).

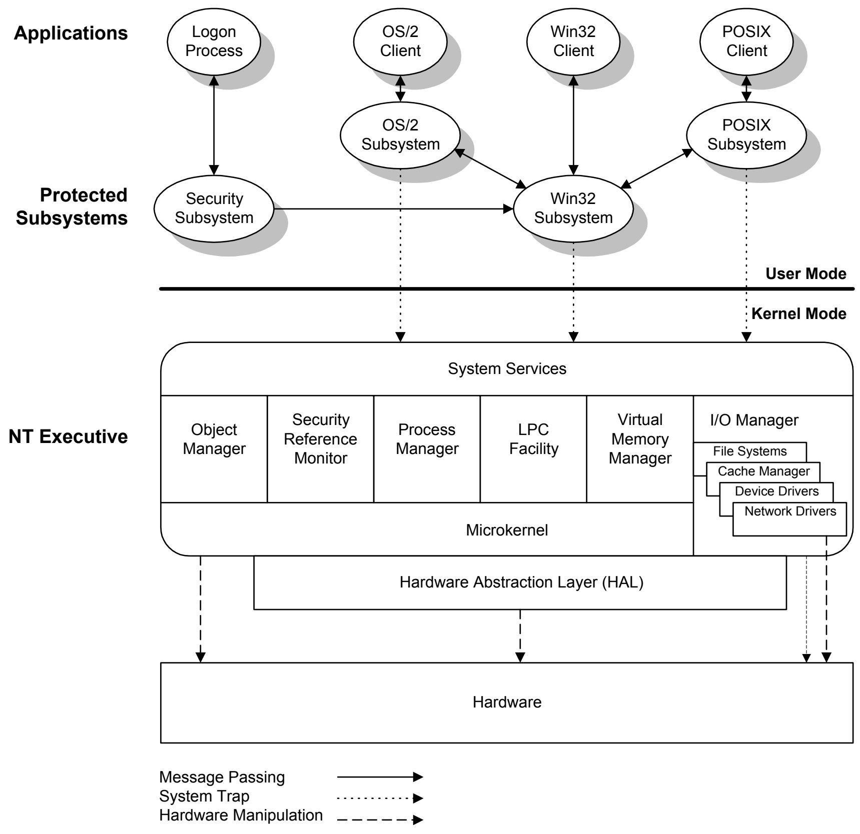

*Figure 1. Windows NT Basic Architecture – version 3.51 and earlier*

In Windows NT 4.0, much of the Win32 graphical user interface (GUI) subsystem—the Window Manager, Graphics Device Interface (GDI), and related graphics drivers—were moved from a body of code that executed in the csrss.exe subsystem process to a kernel mode device driver (win32k.sys).. The console, shutdown, and hard error handling portions remain in user mode. This change significantly improves system performance while decreasing memory requirements, and has no impact on application developers. Applications now access the GUI implementation subsystems just as they access other system services, such as I/O and memory management. This change only serves to demonstrate the maintainability and flexibility of the Windows NT modular design.

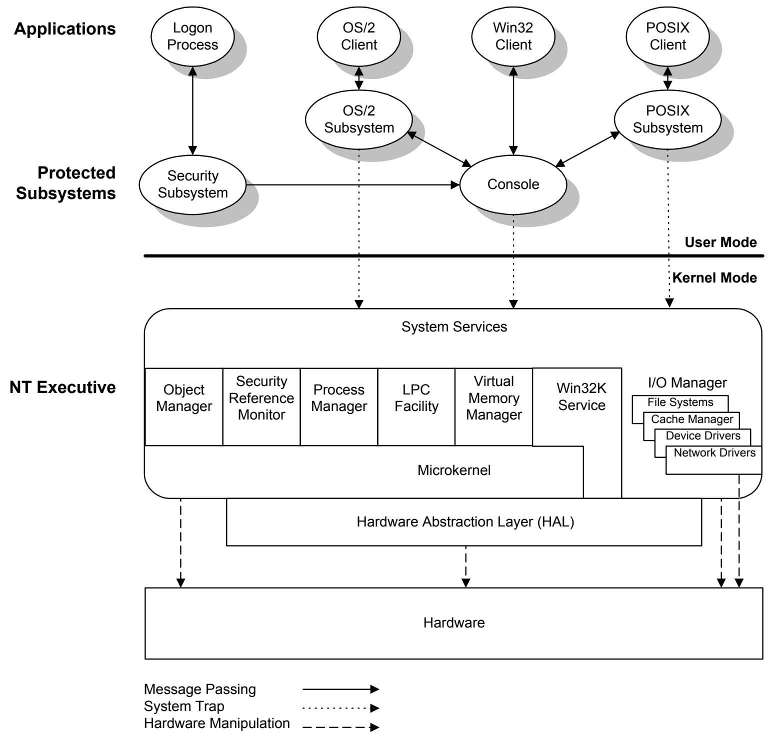

*Figure 2. Windows NT Basic Architecture – version 4.0*

## Built-in Robustness

Windows NT meets its primary goal of robustness as follows:

- The kernel mode portion of the system exports well-defined APIs that, in general, do not have mode parameters or other magic flags. Therefore, the APIs are simple to implement, easy to test, and easy to document.
- A formal design document was produced for all portions of the Windows NT system prior to coding. This effort led to well-documented interfaces for native services and internal functions.
- The partitioning of major components, such as Win32, OS/2, and POSIX, into separate subsystems resulted in a simple, elegant designs for these subsystems. Each subsystem implements only those features needed to provide its API set.
- The prevalent use of frame-based exception handlers (exception handlers associated with a particular procedure or part of a procedure) allows Windows NT and its subsystems to catch programming errors and filter bad or inaccessible parameters in an efficient and reliable manner.

The division of the operating system into kernel-mode system services and subsystems adds a layer of validation to ensure that poorly behaved applications can't crash the operating system.

The next section of this document describes the argument probing and capture requirements to which all system services must adhere. The requirements described here are part of a living document that has existed since the project began in 1989, and serve to illustrate Microsoft’s longstanding commitment to robustness and maintainability in the Windows NT code base.

## ARGUMENT VALIDATION IN WINDOWS NT

Since its inception, Windows NT development has required that system services be written to be robust and provide protection against malicious attack and inadvertent program bugs. To meet the goal of robustness, it must not be possible to crash or corrupt the system by passing an invalid argument value, a pointer to memory that is not accessible to the caller, or by dynamically altering or deleting the memory occupied by an argument in a simultaneously executing thread.

The next few subsections detail some of the standards and coding practices used in the development of past and future versions of Windows NT.

### System Service Requirement

To ensure robust system operation, each system service must ensure that the arguments on which it operates are valid (that is, the values are correct). This requires the service to capture the values and probe the argument addresses at appropriate points. In general, a system service should capture all arguments at the outset. This ensures that the caller or one of its threads cannot dynamically alter the value of the argument after it has been read and verified, or delete the memory in which it is contained.

In some cases, it is not necessary to capture the value of an argument immediately. Such is the case for I/O buffers and name strings. However, all pointers MUST be captured and the addresses to which they point MUST be probed for accessibility.

Fortunately, most arguments do not need explicit capture since they are passed in registers. Arguments that are passed in memory are probed and captured by the system service dispatcher as necessary.

### Address Space Layout

The address space layout of Windows NT clearly separates user address space from system address space.

- All addresses above the boundary are system addresses and all addresses below the boundary are user addresses. Furthermore, at the boundary between user address space and system address space, there is a 64K barrier that is inaccessible to both modes. With this address space design, it is possible to determine whether an address is a valid user address simply by comparing boundaries.
- Pages in the system part of the address space are owned by kernel mode and are not accessible to the user unless they are double-mapped into the user portion of the address space. Pages in the user part of the address space are owned by user mode. The executive *never* creates a page that is owned by kernel mode in the user part of the address space.

## System Service Operation

System service operation should occur as described in the following paragraphs.

When a system service is called, the trap handler gets control, saves state, and transfers control to the system service dispatcher. The system service dispatcher determines which system service is being called, and obtains the address of the appropriate function and the number of in-memory arguments from a dispatch table.

If the previous processor mode is **user** and there is one or more in-memory arguments, the in-memory argument list is probed and copied to the kernel stack.

- If an access violation occurs during the copy, the system service finishes with a status of “Access Violation.”
- If an access violation does not occur, the pointer to the in-memory argument list is changed to point to the copy of the arguments on the kernel stack.

The system service dispatcher sets up a catch-all condition handler and then calls the system service function.

The system service establishes an exception handler. This handler should handle any access violation that may occur as argument pointers are de-referenced to read or write actual argument values.

The system service obtains the previous processor mode.

- If the previous mode was **kernel**, there is no need to probe any arguments. The executive does not call one of its own services and provide bad arguments.
- If the previous mode was **user**, any argument values that are read or written by de-referencing a pointer must be probed for accessibility. To probe a pointer, the service first ensures that the address of the variable is within the user's address space, and then reads or writes the variable as appropriate. The code that actually probes pointer-related arguments does not set up a condition handler. It merely does a boundary check and then reads or writes the indicated location. If the boundary check fails or if the memory is inaccessible, an access violation occurs. (Note that probes are not expensive in terms of system resources.)

The code at the beginning of a system service should be constructed as follows:

```
PreviousMode = KeGetPreviousMode();
if (PreviousMode != KernelMode) {
  try {
    ProbeForWrite(ProcessInformation,
           ProcessInformationLength,
           sizeof(ULONG));
    if (ARGUMENT_PRESENT(ReturnLength)) {
      ProbeForWriteUlong(ReturnLength);
    }
  } except(EXCEPTION_EXECUTE_HANDLER) {
    return GetExceptionCode();
```

```
        }
    }
```

This code sequence guarantees that all address parameters that the service is going to write through are valid user-mode addresses. If any of the probes were to fail, an exception would occur and the exception code would be returned as the service status. Since for this service, **ProcessInformation**, and **ReturnLength** are direct arguments to the service, they do not need to be captured manually inside the service. The system service dispatcher captures them on entry to the system.

## CONCLUSION

From the outset, Windows NT was designed to be a robust, reliable operating system that could be easily maintained and that could be extended to take advantage of new technologies as they were developed. The system includes a highly functional executive that executes in kernel mode, and provides native system services. The executive provides the sole, secure entry point into the system—there are no back door entry points that could compromise security or damage the system in any way. In addition, the design includes a layer of protected system services that function in user mode between the application layer and the operating system. This modular approach allows additional crucial services to be added—with no change to the executive layer. Each major executive subsystem has been extensively documented to ensure that standard coding practices are used and that all features adhere to the system design and are maintainable over time.

This design has remained virtually unchanged from 1989 until now. It provides the foundation for all versions of Windows NT to date, including the highly distributed version that will be shipped in 1998.

## FOR MORE INFORMATION

For more information on the design and architecture of Windows NT, refer to the product documentation. Historical information about the development and early design of Windows NT can be found in *Inside Windows NT* from Microsoft Press.

For the latest information on Windows NT Server, check out our World Wide Web site at http://www.microsoft.com/ntserver the Windows NT Server Forum on the Microsoft Network (GO WORD: MSNTS).